Atty Docket No.: 02-5604 (4028-03000)

Express Mail Label No.

EV333400295US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR UNITED STATES LETTERS PATENT

DATA PROCESSING SYSTEMS INCLUDING HIGH PERFORMANCE BUSES AND INTERFACES, AND ASSOCIATED COMMUNICATION METHODS

By:

Hung T. Nguyen
4632 Portrait Lane
Plano, TX 75024

Citizenship: USA

Shannon A. Wichman
1612 Watersedge Drive
McKinney, TX 75070

Citizenship: USA

DATA PROCESSING SYSTEMS INCLUDING HIGH PERFORMANCE BUSES AND INTERFACES, AND ASSOCIATED COMMUNICATION METHODS

FIELD OF THE INVENTION

[0001] This invention relates generally to data processing systems and, more particularly, to data processing systems including devices coupled via buses.

BACKGROUND OF THE INVENTION

[0002] A typical data processing system includes a processor coupled to one or more devices. The devices may include, for example, peripheral devices. Peripheral devices typically have specific functions, and are often input/output (I/O) devices. For example, a typical personal computer (PC) system includes a processor coupled to a monitor, a mouse, a keyboard, and a printer. The monitor and the printer are output devices, while the mouse and the keyboard are input devices. A compact disk (CD) read-write drive is an example of a peripheral device that is both an input device and an output device.

[0003] A coprocessor is a special purpose processing unit that assists a processor in performing certain types of operations, particularly computationally demanding operations. For example, a data processing system may include a processor coupled to a math (numeric) coprocessor, wherein the math coprocessor performs certain mathematical computations, particularly floating-point operations. In addition to math coprocessors, graphics coprocessors for manipulating graphic images are also common.

[0004] In known data processing systems including processors coupled to coprocessors, the processor executes instructions from one instruction set (e.g., processor instructions of a processor instruction set), and the coprocessor executes instructions from another instruction set (e.g., coprocessor instructions of a coprocessor instruction set). Due to the special purpose nature of coprocessors, the processor and coprocessor instruction sets typically differ substantially, and are defined by manufacturers of the processor and coprocessor, respectively.

[0005] To take advantage of the coprocessor, software programs must be written to include coprocessor instructions of the coprocessor instruction set. When the processor is executing

instructions of a software program and encounters a coprocessor instruction, the processor issues the coprocessor instruction to the coprocessor. The coprocessor executes the coprocessor instruction, and typically returns a result to the processor.

[0006] A bus is a set of wires, lines or connections used to transfer signals. Buses are commonly used to transfer data between components of electronic systems such as data processing systems. For example, the typical PC system includes a higher speed local bus to accommodate higher speed components (e.g., the monitor) and a lower speed expansion bus for lower speed devices (e.g., the mouse, the keyboard, and the printer).

[0007] Many modern processors employ a technique called pipelining to execute more software program instructions (instructions) per unit of time. In general, processor execution of an instruction involves fetching the instruction (e.g., from a memory system), decoding the instruction, obtaining needed operands, using the operands to perform an operation specified by the instruction, and saving a result. In a pipelined processor, the various steps of instruction execution are performed by independent units called pipeline stages. In the pipeline stages, corresponding steps of instruction execution are performed on different instructions independently, and intermediate results are passed to successive stages. By permitting the processor to overlap the executions of multiple instructions, pipelining allows the processor to execute more instructions per unit of time.

[0008] In general, a "scalar" processor issues instructions for execution one at a time, and a "superscalar" processor is capable of issuing multiple instructions for execution at the same time. A pipelined scalar processor concurrently executes multiple instructions in different pipeline stages; the executions of the multiple instructions are overlapped as described above. A pipelined superscalar processor, on the other hand, concurrently executes multiple instructions in different pipeline stages, and is also capable of concurrently executing multiple instructions in the same pipeline stage.

[0009] As used herein, the term "interrupt request signal," or simply "interrupt signal," refers to a control signal which indicates a high-priority request for service. For example, a peripheral device connected to a processor may assert an interrupt signal when ready to transmit data to the processor, or to receive data from the processor. It is noted that an interrupt signal generated external to a processor may not be synchronized with a clock signal of the processor.

[0010] The two general categories of types of interrupt signals are "non-maskable" and "maskable." The typical processor described above also has a non-maskable interrupt (NMI) terminal for receiving an NMI signal, and a maskable interrupt (IRQ) terminal for receiving an IRQ signal. The NMI signal is typically asserted when a catastrophic event has occurred or is about to occur. Examples of non-maskable interrupts include bus parity error, failure of a critical hardware component such as a timer, and imminent loss of electrical power.

[0011] In general, maskable interrupts are lower-priority requests for service that need not be tended to immediately. Maskable interrupts may be ignored by the processor under program control. A request for service from a peripheral device which is ready to transmit data to a processor, or receive data from the processor, is an example of a maskable interrupt. An interrupt controller (e.g., a programmable interrupt controller or PIC) connected to the processor typically receives maskable interrupt requests from devices connected to the processor, and prioritizes the interrupt requests.

[0012] When a processor receives an interrupt, application program execution stops, the contents of certain critical registers are saved (e.g., the internal state of the processor is saved), and internal control is transferred to an interrupt service routine (e.g., an interrupt handler) which corresponds to the type of interrupt received. In the case of a maskable or non-maskable interrupt, the interrupt controller typically identifies the interrupt to be serviced.

[0013] In a vectored interrupt system, the interrupt controller typically provides a number or instruction address assigned to the interrupt to an instruction sequencing module of the processor (e.g., during an interrupt acknowledge operation). A non-maskable interrupt is typically assigned a specific interrupt number. The processor uses the interrupt number as an index into the interrupt vector table to obtain the address of the appropriate interrupt service routine. When the interrupt service routine is completed, the saved contents of the critical registers are restored (e.g., the state of the processor is restored), and the processor resumes application program execution at the point where execution was interrupted.

SUMMARY OF THE INVENTION

[0014] A processor is disclosed that executes an instruction including a user-defined value, wherein the user-defined value is either an address or a command, and provides the user-defined value during execution of the instruction. In one embodiment the processor includes a bus interface adapted for coupling to a bus having multiple signal lines. The processor drives the

user-defined address or command upon one or more signal lines of the bus via the bus interface during execution of the instruction.

[0015] A data processing system is described including the above described processor coupled to a device including an addressable register. The device receives the user-defined address from the processor and accesses the addressable register in response to the user-defined address.

[0016] A method is disclosed for obtaining a value stored in an addressable register. The method involves driving an address of the addressable register on multiple address signal lines of a bus, and an asserted read control signal on a read control signal line of the bus, during a first stage of an instruction execution pipeline. The value is received via multiple data signal lines of the bus when a corresponding ready signal driven on a ready signal line of the bus is asserted during a second stage of the instruction execution pipeline subsequent to the first stage.

[0017] A method is described for providing a value stored in an addressable register. The method involves receiving an address driven on multiple address signal lines of a bus when a read control signal driven on a read control signal line of the bus is asserted during a first stage of an instruction execution pipeline. If the address is an address of the addressable register, the contents of the addressable register are driven on multiple data signal lines of the bus, and an asserted ready signal on a ready signal line of the bus, during a second stage of the instruction execution pipeline subsequent to the first stage.

[0018] A method is disclosed for storing a value in an addressable register. The method involves driving an address of the addressable register on multiple address signal lines of a bus, and an asserted write control signal on a write control signal line of the bus, during a first stage of an instruction execution pipeline. The value to be stored in the addressable register is driven on multiple data signal lines of the bus, and an asserted ready signal on a ready signal line of the bus, during a second stage of the instruction execution pipeline subsequent to the first stage.

[0019] Another method is described for storing a value in an addressable register. The method involves receiving an address driven on multiple address signal lines of a bus when a write control signal driven on a write control signal line of the bus is asserted during a first stage of an instruction execution pipeline. If the address is an address of the addressable register, the value is received via multiple data signal lines of the bus when a corresponding ready signal driven on a ready signal line of the peripheral bus is asserted during a second stage of the

instruction execution pipeline subsequent to the first stage. The value is stored in the addressable register.

[0020] A method is disclosed for modifying a value stored in an addressable register. The method involves driving an address of the addressable register on multiple address signal lines of a bus, an asserted read control signal on a read control signal line of the bus, and an asserted write control signal on a write control signal line of the bus during a first stage of an instruction execution pipeline. The value is received via a first set of data signal lines of the bus when a corresponding ready signal driven on a ready signal line of the bus is asserted during a second stage of the instruction execution pipeline subsequent to the first stage. The value is modified during a third stage of the instruction execution pipeline subsequent to the second stage. The modified value is driven on a second set of data signal lines of the bus, and an asserted ready signal on a ready signal line of the bus, during a fourth stage of the instruction execution pipeline subsequent to the third stage.

BRIEF DESCRIPTION OF THE DRAWINGS

[0021] The invention may be understood by reference to the following description taken in conjunction with the accompanying drawings, in which like reference numerals identify similar elements, and in which:

[0022] Fig. 1 is a diagram of one embodiment of a data processing system including a processor coupled to a device via a device bus, and to a memory system;

[0023] Fig. 2 is a diagram of one embodiment of data processing system of Fig. 1 wherein the processor is coupled to a peripheral system via a peripheral bus, and wherein the memory system stores a peripheral instruction, and wherein the processor includes a peripheral bus interface adapted for coupling to signal lines of the peripheral bus, and wherein the peripheral bus conveys an INTERRUPT signal;

[0024] Fig. 3 is a diagram of one embodiment of the processor of Fig. 2, wherein the processor includes an instruction sequencing unit and a load/store unit;

[0025] Fig. 4 is a diagram of one embodiment of the peripheral system of Fig. 2, wherein the peripheral system includes multiple peripheral devices;

[0026] Fig. 5 is a diagram illustrating a representative one of the peripheral devices of Fig. 4, wherein the representative peripheral device includes an addressable register having an address within an address space of the processor of Figs. 2 and 3;

[0027] Fig. 6 is a diagram illustrating one embodiment of an instruction execution pipeline implemented within the processor of Figs. 2 and 3;

[0028] Figs. 7A-7B illustrate exemplary embodiments of the peripheral instruction of Fig. 2;

[0029] Fig. 8 is a diagram illustrating component signals of the INTERRUPT signal of Fig. 2 and logic of one embodiment of the instruction sequencing unit of Fig. 3;

[0030] Fig. 9 is a diagram illustrating logic within the embodiment of the instruction sequencing unit of Fig. 8 and embodiments of the peripheral bus interface of Fig. 2 and the load/store unit of Fig. 3;

[0031] Fig. 10 is a timing diagram depicting voltages of signals driven on the peripheral bus of Fig. 2 versus time during an exemplary read transaction;

[0032] Fig. 11 is a timing diagram depicting voltages of signals driven on the peripheral bus of Fig. 2 versus time during another read transaction;

[0033] Fig. 12 is a flow chart of one embodiment of a method for obtaining a value stored in an addressable register;

[0034] Fig. 13 is a flow chart of one embodiment of a method for providing a value stored in an addressable register;

[0035] Fig. 14 is a timing diagram depicting voltages of signals driven on the peripheral bus of Fig. 2 versus time during an exemplary write transaction;

[0036] Fig. 15 is a timing diagram depicting voltages of signals driven on the peripheral bus of Fig. 2 versus time during another write transaction;

[0037] Fig. 16 is a flow chart of one embodiment of a method for storing a value in an addressable register;

[0038] Fig. 17 is a flow chart of one embodiment of another method for storing a value in an addressable register;

[0039] Fig. 18 is a timing diagram depicting voltages of signals driven on the peripheral bus of Fig. 2 versus time during an exemplary read-modify-write transaction;

[0040] Figs. 19A and 19B in combination form a flow chart of one embodiment of a method for modifying a value stored in an addressable register;

[0041] Fig. 20 is a timing diagram depicting voltages of signals driven on the peripheral bus of Fig. 2 versus time during an exemplary interrupt request;

[0042] Fig. 21 is a timing diagram depicting voltages of signals driven on the peripheral bus of Fig. 2 versus time during an exemplary nested interrupt request;

[0043] Fig. 22 is a timing diagram depicting voltages of signals driven on the peripheral bus of Fig. 2 versus time during another nested interrupt request;

[0044] Figs. 23A and 23B in combination form a flow of one embodiment of a method for handling an interrupt request;

[0045] Fig. 24 is a diagram of another embodiment of the data processing system of Fig. 1 wherein the processor is coupled to a coprocessor via a coprocessor bus, and wherein the memory system stores a coprocessor instruction;

[0046] Fig. 25 is a diagram of one embodiment of the processor of Fig. 24;

[0047] Figs. 26A-27C illustrate exemplary embodiments of the coprocessor instruction of Fig. 24;

[0048] Fig. 28 is a diagram illustrating how operations of the coprocessor of Fig. 24 are synchronized with operations of the processor of Figs. 24 and 25 during execution of the coprocessor instruction of Fig. 24; and

[0049] Fig. 29 is a diagram of one embodiment of the data processing system of Fig. 24 wherein the processor and the coprocessor are loosely coupled.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0050] In the following disclosure, numerous specific details are set forth to provide a thorough understanding of the present invention. However, those skilled in the art will appreciate that the present invention may be practiced without such specific details. In other instances, well-known elements have been illustrated in schematic or block diagram form in order not to obscure the present invention in unnecessary detail. Additionally, for the most part, details concerning network communications, electromagnetic signaling techniques, and the like, have been omitted inasmuch as such details are not considered necessary to obtain a complete understanding of the present invention, and are considered to be within the understanding of persons of ordinary skill in the relevant art. It is further noted that all functions described herein may be performed in either hardware or software, or a combination thereof, unless indicated otherwise. Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, components may be referred to by different names. This document does not intend to distinguish between

components that differ in name, but not function. In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "including, but not limited to...". Also, the term "couple" or "couples" is intended to mean either an indirect or direct electrical or communicative connection. Thus, if a first device couples to a second device, that connection may be through a direct connection, or through an indirect connection via other devices and connections.

[0051] Fig. 1 is a diagram of one embodiment of a data processing system 100 including a processor 102 coupled to, and in communication with, a device 104 via a device bus 106. The device bus 106 includes multiple signal lines for conveying signals between the processor 102 and the device 104. In general, the processor 102 and the device 104 cooperate to achieve a desired result. Embodiments of the processor 102 and the device 104 are described below. The device 104 may be, for example, a peripheral system including one or more peripheral devices. Alternately, the device 104 may be a coprocessor that extends or augments a computational capability of the processor 102.

[0052] In the embodiment of Fig. 1, the processor 102 includes a device bus interface 108 adapted for coupling to the signal lines of the device bus 106. In the embodiment of Fig. 1, the signal signals conveyed between the processor 102 and the device 104 via the device bus 106 include an n-bit "ADDRESS/COMMAND" signal ($n > 1$), a first 1-bit "READY/VALID" signal, a first n-bit "DATA" signal, n "CONTROL" signals, a second n-bit "DATA" signal, a second 1-bit "READY/VALID" signal, and an m-bit "INTERRUPT" signal ($m \geq 1$).

[0053] In general, the n-bit ADDRESS/COMMAND signal is used to convey an n-bit address or an n-bit command from the processor 102 to the device 104. In some embodiments, the n-bit ADDRESS/COMMAND signal is used to convey an n-bit address from the processor 102 to the device 104. As described below, the n-bit address may be, for example, an address of an addressable register of the device 104. In general, an "addressable" register has a corresponding value known as an address and is accessed via the address. In general, a hardware address includes multiple ordered bits. Addresses of addressable registers may be assigned by a user.

[0054] In other embodiments, the n-bit ADDRESS/COMMAND signal is used to convey an n-bit, user-defined command from the processor 102 to the device 104. In general, the user-defined command includes multiple ordered bits, wherein the values of the bits are assigned by a user. The

device 104 may be configured to interpret the user-defined command specified by the *n*-bit COMMAND signal, and to perform a corresponding function.

[0055] In some embodiments, the first READY/VALID signal corresponds to the *n*-bit ADDRESS/COMMAND signal, and indicates whether the *n*-bit ADDRESS/COMMAND signal is valid. In other embodiments, the first READY/VALID signal corresponds to the first *n*-bit DATA signal, and indicates whether the *n*-bit DATA signal is valid.

[0056] The first *n*-bit DATA signal is used to convey *n* bits of data from the processor 102 to the device 104, and the second *n*-bit DATA signal is used to convey *n* bits of data from the device 104 to the processor 102. Such uni-directional signals simplify the device bus interface 108.

[0057] For example, when the first READY/VALID signal corresponds to the first *n*-bit DATA signal and the first READY/VALID signal and a write control signal are asserted, the peripheral system 202 may respond by writing data conveyed by the first *n*-bit DATA signal to an addressable register of the peripheral system 202 having the address specified by the *n*-bit ADDRESS/COMMAND signal.

[0058] Alternately, in response to a valid *n*-bit ADDRESS/COMMAND signal conveying an *n*-bit user-defined command, the device 104 may perform a function on data conveyed by the first *n*-bit DATA signal, thereby producing a result. The device 104 may convey the result to the processor 102 via the second *n*-bit DATA signal.

[0059] The second READY/VALID signal corresponds to the second *n*-bit DATA signal, and indicates whether the second *n*-bit DATA signal is valid. In general, the *m*-bit INTERRUPT signal conveys interrupt request information. As indicated in Fig. 1, the device interface bus 108 may or may not receive the *m*-bit INTERRUPT signal from the device 104 via the device bus 106. As described below, in some embodiments of the data processing system 100, an interrupt control unit may reside in the device 104. In this situation, *m* may be greater than 1, and the device interface bus 108 may receive the multiple INTERRUPT signals from the device 104. In other embodiments of the data processing system 100, the interrupt control unit may reside in the processor 102, the device 104 may assert a single INTERRUPT signal when the device 104 needs service (e.g., when the device 104 needs to communicate with the processor 102), and the device interface bus 108 may not receive the single INTERRUPT signal from the device 104.

[0060] In the embodiment of Fig. 1, the device bus interface 108 of the processor 102 drives the *n*-bit ADDRESS/COMMAND signal, the first READY/VALID signal, the first *n*-bit DATA

signal, and the n CONTROL signals on corresponding signal lines of the device bus 106, and receives the second n-bit DATA signal and the second READY/VALID signal via corresponding signal lines of the device bus 106. The device 104 receives the n-bit ADDRESS/COMMAND signal, the first READY/VALID signal, the first n-bit DATA signal, and the n CONTROL signals via the corresponding signal lines of the device bus 106, and drives the second n-bit DATA signal and the second READY/VALID signal on the corresponding signal lines of the device bus 106.

[0061] In the embodiment of Fig. 1, the processor 102 is coupled to a memory system 110. In general, the processor 102 fetches and executes instructions of a predefined instruction set stored in the memory system 110. As illustrated in Fig. 1, the memory system 110 includes a software program (i.e., code) 112 including instructions from the instruction set. The code 112 includes a device instruction 114 of the instruction set. In general, execution of the device instruction 114 by the processor 102 causes the processor 102 to communicate with (i.e., send data to and/or receive data from) the device 104.

[0062] For example, as described below, the device instruction 114 may include address information. During execution of the device instruction 114, the processor 102 may read data from and/or write data to an addressable register of the device 104 specified by the address information. Alternately, the device instruction 114 may include a user-defined command. During execution of the device instruction 114, the processor 102 may provide the user-defined command to the device 104. In response to the user-defined command, the device 104 may perform a predetermined function.

[0063] As indicated in Fig. 1, the device 104 may be coupled to the memory system 110, and may access the memory system 110 directly (e.g., in "loosely coupled" embodiments of the data processing system 100). Alternately, the device 104 may not be coupled to the memory system 110, and may depend on the processor 102 to access the memory system 110 and to provide data from the memory system 110 to the device 104 (e.g., in "tightly coupled" embodiments of the data processing system 100).

[0064] Fig. 2 is a diagram of one embodiment of a data processing system 200, wherein the data processing system 200 is one embodiment of the data processing system 100 of Fig. 1. In the data processing system 200, the processor 102 is coupled to, and in communication with, a peripheral system 202 via a peripheral bus 204. In general, the processor 102 and the peripheral system 202 cooperate to achieve a desired result, and the peripheral system 202 includes one or

more peripheral devices. Embodiments of the processor 102 and the peripheral system 202 are described below.

[0065] In the embodiment of Fig. 2, the processor 102 includes a peripheral bus interface 206 adapted for coupling to signal lines of the peripheral bus 204. In the embodiment of Fig. 2, signal lines of the peripheral bus 204 are used to convey several signals between the processor 102 and the peripheral system 202, including an 24-bit address signal "PADR,", a 16-bit data signal "CDATA," a 1-bit ready signal "CDRDY," a 1-bit read control signal "PRD," a 1-bit write control signal "PWR," a 16-bit data signal "PDATA," a 1-bit ready signal "PDRDY," and a 20-bit interrupt signal "INTERRUPT."

[0066] In the embodiment of Fig. 2, the PADR signal is used to convey an 24-bit address from the processor 102 to the peripheral system 202. The 24-bit address may be, for example, the address of an addressable register of the peripheral system 202 as described below. The 16-bit CDATA signal is used to convey 16 bits of data from the processor 102 to the peripheral system 202. The CDRDY signal corresponds to the CDATA signal, and indicates whether the CDATA signal is valid.

[0067] The read control signal PRD is asserted when the peripheral system 202 is to perform a read operation, and the write control signal PWR is asserted when the peripheral system 202 is to perform a write operation. The 16-bit PDATA signal is used to convey 16 bits of data from the peripheral system 202 to the processor 102. Again, uni-directional signals simplify the peripheral bus interface 206. The ready signal PDRDY corresponds to the PDATA signal, and indicates whether the PDATA signal is valid.

[0068] For example, when the write control signal PWR and the ready signal CDRDY are asserted, the peripheral system 202 may respond by writing data conveyed by the CDATA signal to an addressable register of the peripheral system 202 having the address specified by the PADR signal.

[0069] In general, the 20-bit INTERRUPT signal conveys interrupt information. In the embodiment of Fig. 2, an interrupt control unit resides in the peripheral system 202, and the device interface bus 108 receives the 20-bit INTERRUPT signal from the peripheral system 202 via the peripheral bus 204. The component signals making up the 20-bit INTERRUPT signal are described below.

[0070] In the embodiment of Fig. 2, the peripheral bus interface 206 of the processor 102 drives the PADR signal, the CDATA signal, the CDRDY signal, the PRD and PWR control signals, and a portion of the component signals making up the INTERRUPT signal on corresponding signal lines of the peripheral bus 204, and receives the PDATA signal, the PDRDY signal, and a remainder of the component signals making up the INTERRUPT signal via corresponding signal lines of the peripheral bus 204. The peripheral system 202 receives the PADR signal, the CDATA signal, the CDRDY signal; the PRD and PWR control signals, and the portion of the component signals making up the INTERRUPT signal via the corresponding signal lines of the peripheral bus 204, and drives the PDATA signal, the PDRDY signal, and the remainder of the component signals making up the INTERRUPT signal on the corresponding signal lines of the peripheral bus 204.

[0071] In the embodiment of Fig. 2, the code 112 stored within the memory system 110 includes a peripheral instruction 208 of the instruction set. In general, execution of the peripheral instruction 208 by the processor 102 causes the processor 102 to communicate with (i.e., send data to and/or receive data from) the peripheral system 202.

[0072] For example, the peripheral instruction 208 may include address information. During execution of the peripheral instruction 206, the processor 102 may read data from or write data to an addressable register of the peripheral system 202 specified by the address information.

[0073] As indicated in Fig. 2, the peripheral system 202 may be coupled to the memory system 110, and may access the memory system 110 directly (e.g., in loosely coupled embodiments of the data processing system 200). Alternately, the peripheral system 202 may not be coupled to the memory system 110, and may depend on the processor 102 to access the memory system 110 and to provide data from the memory system 110 to the peripheral system 202 (e.g., in tightly coupled embodiments of the data processing system 200).

[0074] Fig. 3 is a diagram of one embodiment of the processor 102 of Fig. 2. As indicated in Fig. 3, the processor 102 receives a clock signal "CLOCK" and executes instructions dependent upon the CLOCK signal. More specifically, the processor 102 includes several functional units described below, and operations performed within the functional units are synchronized by the CLOCK signal.

[0075] In the embodiment of Fig. 3, in addition to the peripheral bus interface 206 of Fig. 2, the processor 102 includes an instruction prefetch unit 300, an instruction sequencing unit 302, a

load/store unit (LSU) 304, an execution unit 306, register files 308, and a pipeline control unit 310. The instruction prefetch unit 300, the instruction sequencing unit 302, the load/store unit (LSU) 304, the execution unit 306, the register files 308, and the pipeline control unit 310 may be considered functional units of the processor 102, and may contain other functional units.

[0076] In the embodiment of Fig. 3, the processor 102 is a pipelined superscalar processor core. That is, the processor 102 implements an instruction execution pipeline including multiple pipeline stages, concurrently executes multiple instructions in different pipeline stages, and is also capable of concurrently executing multiple instructions in the same pipeline stage.

[0077] In general, the instruction prefetch unit 300 fetches instructions from the memory system 110 of Fig. 1, and provides the fetched instructions to the instruction sequencing unit 302. In one embodiment, the instruction prefetch unit 300 is capable of fetching up to 8 instructions at a time from the memory system 110, partially decodes and aligns the instructions, and stores the partially decoded and aligned instructions in an instruction cache within the instruction prefetch unit 300.

[0078] The instruction sequencing unit 302 receives (or retrieves) partially decoded instructions from the instruction cache of the instruction prefetch unit 300, fully decodes the instructions, and stores the fully decoded instructions in an instruction queue. In one embodiment, the instruction sequencing unit 302 is capable of receiving (or retrieving) multiple partially decoded instructions from the instruction cache of the instruction prefetch unit 300, and decoding the multiple partially decoded instructions, during a single cycle of the CLOCK signal.

[0079] In one embodiment, the instruction sequencing unit 302 translates instruction operation codes (i.e., opcodes) into native opcodes for the processor. The instruction sequencing unit 302 checks the multiple decoded instructions using grouping and dependency rules, and provides (i.e., issues) one or more of the decoded instructions conforming to the grouping and dependency rules as a group to the to the load/store unit (LSU) 304 and/or the execution unit 306 for simultaneous execution.

[0080] The load/store unit (LSU) 304 is used to transfer data between the processor 102 and the memory system 110. In one embodiment, the load/store unit (LSU) 304 includes 2 independent load/store units. Each of the 2 independent load/store units accesses the memory system 110 via separate load/store buses, and includes a separate address generation unit (AGU)

for generating and translating address signals needed to access values stored in the memory system 110.

[0081] The execution unit 306 is used to perform operations specified by instructions (and corresponding decoded instructions). In one embodiment, the execution unit 306 includes 2 independent arithmetic logic units (ALUs), and 2 independent multiply/accumulate units (MAUs).

[0082] In general, the register files 308 include one or more register files of the processor 102. In one embodiment, the register files 308 includes an address register file and a general purpose register file. The address register file includes 8 32-bit address registers, and the general purpose register file includes 16 16-bit general purpose registers. The 16 16-bit registers of the general purpose register file can be paired to form 8 32-bit general purpose registers. The registers of the register files 308 may, for example, be accessed via read/write enable signals from the pipeline control unit 310.

[0083] In general, the pipeline control unit 310 controls an instruction execution pipeline implemented within the processor 102 and described in more detail below.

[0084] Fig. 4 is a diagram of one embodiment of the peripheral system 202 of Fig. 2. In the embodiment of Fig. 4, the peripheral system 202 includes an interrupt control unit 400, a data management unit 402, and multiple peripheral devices represented by peripheral devices 404A and 404B. The peripheral device 404A may be, for example, a timer, a serial port, or a parallel port. Similarly, the peripheral device 404B may be, for example, a timer, a serial port, or a parallel port. Herein below, the multiple peripheral devices represented by peripheral devices 404A and 404B will be referred to collectively as the peripheral devices 404.

[0085] As indicated in Fig. 4, the PADR signal, the CDATA signal, the CDRDY signal, the PRD and PWR control signals driven on the peripheral bus 204 by the peripheral bus interface 206 of the processor 102 are received by the interrupt control unit 400, the data management unit 402, and the peripheral devices 404. The interrupt control unit 400, the data management unit 402, and the peripheral devices 404 respond to the PADR signal, the CDATA signal, the CDRDY signal, the PRD and PWR control signals independently.

[0086] The data management unit 402 is coupled to each of the peripheral devices 404 via a peripheral read data bus 406, and receives read data from the peripheral devices 404 via a peripheral read data bus 406. The data management unit 402 is also coupled to the interrupt control unit 400 and receives read data from the interrupt control unit 400. The data management

unit 402 uses the received read data to generate the PDATA and PDRDY signals, and drives the PDATA and PDRDY signals on the corresponding signal lines of the peripheral bus 204.

[0087] The interrupt control unit 400 is coupled to each of the peripheral devices 404 via a peripheral interrupt bus 408, and receives interrupt signals from the peripheral devices 404 via a peripheral interrupt bus 406. The interrupt control unit 400 is also coupled to the data management unit 402, and receives an interrupt signal from the data management unit 402. The interrupt control unit 400 also receives component signals making up the INTERRUPT signal from the processor 102 via the peripheral bus 204. The component signals making up the INTERRUPT signal are described below. In general, and as described below, the interrupt control unit 400 uses the received interrupt signals and the component signals making up the INTERRUPT signal received from the processor 102 to generate a portion of the component signals making up the INTERRUPT signal, and drives the portion of the component signals making up the INTERRUPT signal on corresponding signal lines of the peripheral bus 204.

[0088] More specifically, the interrupt control unit 400 helps to implement a vectored priority interrupt system in the data processing system 200 of Fig. 2 in which higher priority interrupts are handled (i.e., serviced) first. A non-maskable interrupt (NMI) signal has the highest priority of all the interrupt signals. In one embodiment, the interrupt control unit 400 includes a 16-bit interrupt request register having bit locations corresponding to 2 non-maskable interrupt signals and 14 maskable interrupt bit locations. The 2 non-maskable interrupt signals include the NMI signal and a device emulation interrupt (DEI) signal. When an interrupt signal is received, the corresponding bit location in the interrupt request register is set to '1'. Each bit location in the interrupt request register is cleared only when the processor 102 services the corresponding interrupt signal, or explicitly by software.

[0089] In one embodiment, the interrupt control unit 400 also includes an interrupt mask register containing mask bit locations for each of the 14 maskable interrupts. A mask bit value of '0' (i.e., a cleared bit) prevents the corresponding interrupt from being serviced (i.e., masks the corresponding interrupt signal). The INTERRUPT signal may be one of the 14 maskable interrupt signals.

[0090] In one embodiment, the interrupt control unit 400 also includes two 16-bit interrupt priority registers. Consecutive bit locations in each of the interrupt priority registers are used to store user-defined priority levels associated with the 14 maskable interrupt signals. Software

programs may write to the bit locations of the interrupt priority registers. User-defined interrupt priorities may range from 0b00 (i.e., decimal '0') to 0b11 (i.e., decimal '3'), with 0b00 being the lowest and 0b11 being the highest. (The NMI signal has a fixed priority level of decimal '5', and the DEI signal has a fixed priority level of decimal '4'.)

[0091] Once the interrupt control unit 400 decides to service an interrupt, the interrupt control unit 400 generates component signals of the INTERRUPT signal, and drives the component signals of the INTERRUPT signal on corresponding signal lines of the peripheral bus 204. As described below, the component signals of the INTERRUPT signal are received by the instruction sequencing unit 302 of the processor 102 of Fig. 2. In response to the component signals of the INTERRUPT signal received from the interrupt control unit 400, the instruction sequencing unit 302 stops grouping and issuing instructions in an interrupted program, and instructions of an interrupt service routine are fetched and executed.

[0092] In the embodiment of Fig. 4 each of the peripheral devices 404 includes at least one addressable register. As described above, an addressable register has a corresponding value known as an address and is accessed via the address. Common types of addressable registers include control registers, status registers, and data registers.

[0093] Fig. 5 is a diagram illustrating a representative one of the peripheral devices 404 of Fig. 4. As indicated in Fig. 5, the representative peripheral device 404 includes an addressable register 500. An address of the addressable register 500 resides in an address space 502 of the processor 102 of Figs. 2 and 3. In general, the address space of a processor is defined by a number of bits in address signals. For example, in the embodiments of Figs. 2 and 3, the address signal PADR is a 24-bit signal. Accordingly, in the embodiment of Fig. 2, the address space 500 of the processor 102 includes 2^{24} (16,777,216) different values (i.e., addresses) ranging from 0 to 2^{24} - 1 (16,777,215).

[0094] Fig. 6 is a diagram illustrating one embodiment of the instruction execution pipeline implemented within the processor 102 of Figs. 2 and 3 and controlled by the pipeline control unit 310 of Fig. 3. The instruction execution pipeline (pipeline) allows overlapped execution of multiple instructions. In the embodiment of Fig. 6, the pipeline includes 8 stages: a fetch/decode (FD) stage, a grouping (GR) stage, an operand read (RD) stage, an address generation (AG) stage, a memory access 0 (M0) stage, a memory access 1 (M1) stage, an execution (EX) stage, and a

write back (WB) stage. As indicated in Fig. 6, operations in each of the 8 pipeline stages are completed during a single cycle of the CLOCK signal.

[0095] Referring to Figs. 2 and 3, the instruction fetch unit 300 fetches several instructions (e.g., up to 8 instructions) from the memory system 110 during the fetch/decode (FD) pipeline stage, partially decodes and aligns the instructions, and provides the partially decoded instructions to the instruction sequencing unit 302. The instruction sequencing unit 302 fully decodes the instructions and stores the fully decoded instructions in an instruction queue (described more fully later). The instruction sequencing unit 302 also translates the opcodes into native opcodes for the processor.

[0096] During the grouping (GR) stage, the instruction sequencing unit 302 checks the multiple decoded instructions using grouping and dependency rules, and passes one or more of the decoded instructions conforming to the grouping and dependency rules on to the read operand (RD) stage as a group. During the read operand (RD) stage, any operand values, and/or values needed for operand address generation, for the group of decoded instructions are obtained from the register files 308.

[0097] During the address generation (AG) stage, any values needed for operand address generation are provided to the load/store unit (LSU) 304, and the load/store unit (LSU) 304 generates internal addresses of any operands located in the memory system 110. During the memory address 0 (M0) stage, the load/store unit (LSU) 304 translates the internal addresses to external memory addresses used within the memory system 110.

[0098] During the memory address 1 (M1) stage, the load/store unit (LSU) 304 uses the external memory addresses to obtain any operands located in the memory system 110. During the execution (EX) stage, the execution unit 306 uses the operands to perform operations specified by the one or more instructions of the group. During a final portion of the execution (EX) stage, valid results (including qualified results of any conditionally executed instructions) are stored in registers of the register files 308.

[0099] During the write back (WB) stage, valid results (including qualified results of any conditionally executed instructions) of store instructions, used to store data in the memory system 110 as described above, are provided to the load/store unit (LSU) 304. Such store instructions are typically used to copy values stored in registers of the register files 308 to memory locations of the memory system 110.

[0100] Figs. 7A-7B illustrate exemplary embodiments of the peripheral instruction 208 of Fig. 2. Fig. 7A is a diagram of one embodiment of the peripheral instruction 208 of Fig. 2 wherein the peripheral instruction 208 includes an opcode field 700, a destination register field 702, and a source register field 704. The opcode field 700 contains a value identifying the instruction as a peripheral instruction directed to the peripheral system 202 of Fig. 2 and specifying the particular peripheral instruction format of Fig. 7A. The destination register field 702 identifies a destination register into which the value is to be saved, and the source register field 704 identifies a source register from which a value is to be obtained. The destination register field 702 and the source register field 704 each specify either: (i) a register of the register files 308, or (ii) an address of an addressable register of the peripheral system 202.

[0101] For example, in the assembly language move instruction 'mov %padr, rY' the value 'padr' is an immediate data value, the source register is the 'rY' general purpose register of the register files 308 of Fig. 2, and the destination register is the addressable register of the peripheral system 202 having address 'padr'. Translation of the assembly language instruction 'mov %padr, rY' expectedly results in a machine language instruction having the format of Fig. 7A.

[0102] Execution of the resulting machine language instruction by the processor 102 of Figs. 2 and 3 expectedly results in the processor 102 initiating a write transaction to the peripheral system 202 of Fig. 2 with the address signal PADR equal the value 'padr', the data signal CDATA equal to a value stored in the 'rY' general purpose register of the register files 308, the write control signal PWR asserted, and the ready signal CDRDY asserted. The peripheral system 202 expectedly responds to the write transaction by writing the data conveyed by the CDATA signal (i.e., the value from the 'rY' general purpose register of the register files 308) to the addressable register having address 'padr' specified by the PADR signal. Such write transactions are described in more detail below.

[0103] Similarly, in the assembly language move instruction 'mov rX, %padr' the source register is the addressable register of the peripheral system 202 having address 'padr' and the destination register is the 'rX' general purpose register of the register files 308 of Fig. 2. Translation of the assembly language instruction 'mov rX, %padr' expectedly results in a machine language instruction having the format of Fig. 7A.

[0104] Execution of the resulting machine language instruction by the processor 102 of Figs. 2 and 3 expectedly results in the processor 102 initiating a read transaction to the peripheral system

202 of Fig. 2 with the address signal PADR equal the value 'padr' and the read control signal PRD asserted. The peripheral system 202 expectedly responds to the read transaction by reading the value stored in the addressable register having address 'padr' specified by the PADR signal, and providing the value to the processor 102 by generating the PDATA signal equal to the value, asserting the PDRDY signal, and driving the PDATA and PDRDY signals on the corresponding signal lines of the peripheral bus 204 of Fig. 2. Such read transactions are described in more detail below.

[0105] Fig. 7B is a diagram of another embodiment of the peripheral instruction 208 of Fig. 2 wherein the peripheral instruction 208 includes an opcode field 706, a source/destination register field 708, and an immediate data field 710. The opcode field 706 contains a value identifying the instruction as a peripheral instruction directed to the peripheral system 202 of Fig. 2 and specifying the particular peripheral instruction format of Fig. 7B. The source/destination register field 708 identifies a register from which an operand value is to be obtained, and to which a result value is to be saved. The source/destination register field 708 specifies either: (i) a register of the register files 308, or (ii) an address of an addressable register of the peripheral system 202.

[0106] For example, in the assembly language bit set instruction 'bits %padr, y' the value 'padr' is an immediate data value, the source/destination register is the addressable register of the peripheral system 202 of Fig. 2 having address 'padr', and the bit number to be set is 'y'. Translation of the assembly language instruction 'bits %padr, y' expectedly results in a machine language instruction having the format of Fig. 7B.

[0107] Execution of the resulting machine language instruction by the processor 102 of Figs. 2 and 3 expectedly results in the processor 102 initiating a read-modify-write transaction to the peripheral system 202 of Fig. 2. During the read-modify-write transaction, the processor obtains the value stored in the addressable register of the peripheral system 202 of Fig. 2 having address 'padr', sets bit 'y' of the value, and provides the modified value to the peripheral system 202. Such read-modify-write transactions are described in more detail below.

[0108] Similarly, translations of a bit clear assembly language instruction 'bitc %padr, y' and a bit invert assembly language instruction 'biti %padr, y' expectedly result in machine language instructions having the format of Fig. 7B. During executions of the resulting machine language instructions, the processor 102 expectedly initiates a read-modify-write transaction to the peripheral system 202 of Fig. 2. During the read-modify-write transactions, the processor obtains

the value stored in the addressable register of the peripheral system 202 of Fig. 2 having address 'padr', modifies bit 'y' of the value, and provides the modified value to the peripheral system 202.

[0109] Fig. 8 is a diagram illustrating component signals of the INTERRUPT signal of Fig. 2 and logic of one embodiment of the instruction sequencing unit 302 of Fig. 3. In the embodiment of Fig. 8, several signals make up the 20-bit INTERRUPT signal, including a 1-bit interrupt request signal "PIRQ," a 1-bit new interrupt request signal "PNIRQ," a 16-bit peripheral interrupt vector signal "PIVECT," a 1-bit interrupt inhibit signal "IRQINH," and a 1-bit interrupt return signal "IRQRET."

[0110] In the embodiment of Fig. 8, the processor 102 generates the interrupt return signal IRQRET and the interrupt inhibit signal IRQINH. The peripheral bus interface 206 of the processor 102 registers the interrupt return signal IRQRET and the interrupt inhibit signal IRQINH and drives the interrupt return signal IRQRET and the interrupt inhibit signal IRQINH on the corresponding signal lines of the peripheral bus 204. The interrupt control unit 400 of Fig. 4 generates the interrupt request signals PIRQ and PNIRQ and the interrupt vector signal PIVECT, and drives the interrupt request signals PIRQ and PNIRQ and the interrupt vector signal PIVECT on the corresponding signal lines of the peripheral bus 204. The peripheral bus interface 206 of the processor 102 registers the received interrupt request signals PIRQ and PNIRQ and interrupt vector signal PIVECT.

[0111] The interrupt request signal PIRQ is asserted by the interrupt control unit 400 of Fig. 4 when a functional unit (e.g., a peripheral device) of the peripheral system 202 of Figs 2 and 4 requires service. In response to an interrupt request signal, the processor 102 executes instructions of an interrupt service routine. The last instruction of each interrupt service routine is an interrupt return instruction. The processor 102 asserts the interrupt return signal IRQRET when the interrupt return instruction is grouped for execution. The processor 102 asserts the interrupt inhibit signal IRQINH to inhibit interrupts.

[0112] The new interrupt request signal PNIRQ is asserted by the interrupt control unit 400 of Fig. 4 when a lower priority interrupt request is received while a higher priority interrupt is being handled by the processor 102. In this situation, the processor 102 services the lower priority interrupt after handling the higher priority interrupt.

[0113] The data processing system 200 of Fig. 2 supports nested interrupts. That is, higher priority interrupts can interrupt service routines of lower priority interrupts. The 16-bit peripheral

interrupt vector signal PIVECT is used to specify a priority of a corresponding interrupt request. When by the interrupt control unit 400 of Fig. 4 asserts the interrupt request signals PIRQ and PNIRQ, the interrupt control unit 400 generates the interrupt vector signal PIVECT with the priority associated with the interrupt request, and drives the interrupt vector signal PIVECT on the corresponding signal lines of the peripheral bus 204.

[0114] As described above, in one embodiment, the interrupt control unit 400 includes two 16-bit interrupt priority registers. Consecutive bit locations in each of the interrupt priority registers are used to store user-defined priority levels associated with the 14 maskable interrupt signals. Software programs may write to the bit locations of the interrupt priority registers. User-defined interrupt priorities may range from 0b00 (i.e., decimal '0') to 0b11 (i.e., decimal '3'), with 0b00 being the lowest and 0b11 being the highest. (The NMI signal has a fixed priority level of decimal '5', and the DEI signal has a fixed priority level of decimal '4'.)

[0115] As indicated in Fig. 8, the component signals of the INTERRUPT signal are generated and received by the instruction sequencing unit 302. In the embodiment of Fig. 8, the instruction sequencing unit 302 includes a program counter (PC) unit 800, a trap program counter (TPC) unit 802, a grouping unit 804, and a dispatch unit 806.

[0116] The program counter (PC) unit 800 stores and maintains a program counter for the processor 102. The program counter is stored in a register, and specifies an address of a next instruction to be fetched in the memory system 110 of Figs. 1 and 2. The next instruction to be fetched may reside, for example, in the code 112 of Figs. 1 and 2.

[0117] The trap program counter (TPC) unit 802 includes multiple registers forming a last-in-first-out (LIFO) stack for storing values of the program counter. The last-in-first-out (LIFO) stack of the trap program counter (TPC) unit 802 allows the data processing system 200 of Fig. 2 to handle nested interrupt requests. Operation of the trap program counter (TPC) unit 802 during interrupt requests is described in detail below.

[0118] In the grouping (GR) stage of the pipeline, fully decoded instructions (e.g., from an instruction queue) are provided to the grouping unit 804. The grouping unit 804 performs dependency checks on the fully decoded instructions by applying a predefined set of dependency rules (e.g., write-after-write, read-after-write, write-after-read, etc.). The set of dependency rules determine which instructions can be grouped together for simultaneous execution (e.g., execution in the same cycle of the CLOCK signal).

[0119] During the instruction decoding process, instruction opcodes are translated to internal representations called "native opcodes." The dispatch unit 806 queues native opcodes and other relevant information such as read control signals and register addresses for use by the execution unit 306 of Fig. 3, the register files 308 of Fig. 3, and/or the load/store unit 304 of Fig. 3 at appropriate times during instruction execution.

[0120] In response to an asserted interrupt request signal PIRQ, the grouping unit 804 stops grouping instructions for simultaneous execution. Instructions fetched by the processor 102 and partially decoded up to and including those in the grouping (GR) stage are flushed. Executions of instructions in the operand read (RD) stage, the address generation (AG) stage, the memory access 0 (M0) stage, the memory access 1 (M1) stage, and the execution (EX) stage are completed normally. Subsequently, the value of the program counter is pushed on the stack of the trap program counter (TPC) unit 802, and the program counter is loaded with an address of a first instruction of an interrupt service routine corresponding to the interrupt request. As a result, instructions of the interrupt service routine are fetched and executed.

[0121] As described above, the last instruction of every interrupt service routine is an interrupt return instruction. When the grouping unit 804 groups the interrupt return instruction for execution, logic of the instruction sequencing unit 302 asserts the interrupt return signal IRQRET. During the subsequent operand read (RD) stage of the execution pipeline, the peripheral bus interface 206 drives the asserted interrupt return signal IRQRET on the corresponding signal line of the peripheral bus 204 of Fig. 2. At the same time, the logic of the instruction sequencing unit 302 pops a saved program counter value from the top of the stack of the trap program counter (TPC) unit 802, and stores the value of the program counter in the register of the program counter (PC) unit 800 reserved for the program counter. As a result, the fetching of instructions of the interrupted program is resumed during the next cycle of the CLOCK signal at the point where the program was interrupted.

[0122] Fig. 9 is a diagram illustrating logic within the embodiment of the instruction sequencing unit 302 of Fig. 8 and embodiments of the peripheral bus interface 206 of Fig. 2 and the load/store unit 304 of Fig. 3. In Fig. 9, the peripheral bus interface 206 includes several registers for storing signals, stall logic 900, and a bit manipulation unit (BMU) 902. As indicated in Fig. 9, the ready signal PDRDY received from the peripheral system 202 of Fig. 2 is first registered within the peripheral bus interface 206, then provided to the stall logic 900. The stall

logic uses the ready signal PDRDY and other stall signals to produce a "CORE_STALL" signal. More specifically, if the ready signal PDRDY is not asserted when expected, the stall logic 900 asserts the CORE_STALL signal. The CORE_STALL signal is distributed to logic within the processor 102. When the CORE_STALL signal is asserted, the execution pipeline implemented within the processor 102 is stalled until the CORE_STALL signal is deasserted.

[0123] The bit manipulation unit (BMU) 902 is used during read-modify-write operations (e.g., bit set instructions, bit clear instructions, and bit invert instructions) specifying an addressable register of the peripheral system 202 of Fig. 2. Exemplary bit manipulation instructions specifying addressable registers of the peripheral system 202 of Fig. 2 are described above with regard to Fig. 7B.

[0124] During a read portion of a read-modify-write transaction carried out in response to a bit manipulation instruction specifying an addressable registers of the peripheral system 202, the bit manipulation unit (BMU) 902 receives the data signal PDATA from the peripheral system 202 as input (e.g., as an operand). The data signal PDATA conveys the value stored in the addressable register of the peripheral system 202. During a modify portion of the read-modify-write transaction, the bit manipulation unit (BMU) 902 carries out the bit manipulation operation specified by the bit manipulation instruction, thereby producing a result. During a write portion of the read-modify-write transaction, the result produced by the bit manipulation unit (BMU) 902 is provided to the peripheral system 202 via the data signal CDATA.

[0125] In Fig. 9 the load/store unit 304 includes several registers for storing values, decode logic 904, an address generation unit 906, and a result first-in-first-out (FIFO) buffer 908. As indicated in Fig. 9, the decode logic 904 receives the native opcodes of instructions grouped for simultaneous execution from the dispatch unit 806 of the instruction sequencing unit 302. The decode logic 904 generates the ready signal CDRDY, the read control signal PRD, the write control signal PWR, and one or more control signals dependent upon the native opcodes.

[0126] The load/store unit 304 first registers the ready signal CDRDY, the read control signal PRD, the write control signal PWR, and the control signals, then provides the control signals to the address generation unit 906, and provides the ready signal CDRDY, the read control signal PRD, and the write control signal PWR to the peripheral bus interface 206.

[0127] The address generation unit 906 receives an "OPERAND #1," an "OPERAND #2," and the controls signals produced by the decode logic 904, and produces a result (e.g., an address

value) dependent upon the OPERAND #1, the OPERAND #2, and the controls signals. As indicated in Fig. 9, logic of the load/store unit 304 may provide the result produced by the address generation unit 906 to the peripheral bus interface 206. In this situation, the peripheral bus interface 206 first registers the result produced by the address generation unit 906, then drives the result produced by the address generation unit 906 on the peripheral bus 204 as the address signal PADR.

[0128] Alternately, the logic of the load/store unit 304 may provide the result produced by the address generation unit 906 to the result first-in-first-out (FIFO) buffer 908 as indicated in Fig. 9. At some later time, the result first-in-first-out (FIFO) buffer 908 may provide the result to the peripheral bus interface 206. In this situation, the peripheral bus interface 206 first registers the result, then drives the result on the peripheral bus 204 as the data signal CDATA.

[0129] As indicated in Fig. 9, the peripheral bus interface 206 first buffers the data signal PDATA received from the peripheral system 202. The peripheral bus interface 206 may provide the data signal PDATA to the load/store unit 304, and the load/store unit 304 may provide the data signal PDATA as a "RESULT" signal to other logic within the processor 102.

[0130] In the embodiment of Fig. 9, the peripheral bus interface 206 includes a peripheral data queue 910 used to store the data signal PDATA while the processor 102 is stalled. The peripheral data queue 910 may include, for example, multiple storage elements operated in a first-in-first-out (FIFO) manner.

[0131] Fig. 10 is a timing diagram depicting voltages of signals driven on the peripheral bus 204 of Fig. 2 versus time during an exemplary read transaction. During the read transaction of Fig. 10, the CORE_STALL signal is not asserted, and the execution pipeline implemented within the processor 102 of Fig. 2 is not stalled.

[0132] In Fig. 10, the address signal PADR and the asserted read control signal PRD are driven on the corresponding signal lines of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9 during the memory read 0 (M0) stage of the execution pipeline. As described above, the address signal PADR specifies an address of an addressable register of the peripheral system 202 of Fig. 2. During the subsequent memory read 1 (M1) stage of the execution pipeline, the data signal PDATA and the asserted ready signal PDRDY are driven on the corresponding signal lines of the peripheral bus 204 by the peripheral system 202 of Fig.2. The data signal PDATA expectedly conveys a value stored in the addressable register of the peripheral system 202.

[0133] Fig. 11 is a timing diagram depicting voltages of signals driven on the peripheral bus 204 of Fig. 2 versus time during another read transaction. During the read transaction of Fig. 11, the CORE_STALL signal is asserted, and the execution pipeline implemented within the processor 102 of Fig. 2 is stalled for one cycle of the CLOCK signal.

[0134] As in Fig. 10, the address signal PADR and the asserted read control signal PRD are driven on the corresponding signal lines of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9 during the memory read 0 (M0) stage of the execution pipeline. As described above, the address signal PADR specifies an address of an addressable register of the peripheral system 202 of Fig. 2.

[0135] In Fig. 11, during the subsequent memory read 1 (M1) stage of the execution pipeline, the data signal PDATA is not driven on the corresponding signal lines of the peripheral bus 204 and the ready signal PDRDY is not asserted by the peripheral system 202 of Fig.2. As a result, the stall logic 900 of Fig. 9 asserts the CORE_STALL signal during the next cycle of the CLOCK signal. As a result, the execution pipeline implemented within the processor 102 of Fig. 2 is stalled as indicated in Fig. 11.

[0136] During the same cycle of the CLOCK signal, the data signal PDATA is driven on the corresponding signal lines of the peripheral bus 204 and the ready signal PDRDY is asserted by the peripheral system 202 of Fig.2. In response, the stall logic 900 of Fig. 9 deasserts the CORE_STALL signal. As a result, the memory read 1 (M1) stage of the execution pipeline is effectively extended by one cycle of the CLOCK signal as indicated in Fig. 11. As described above, the data signal PDATA expectedly conveys a value stored in the addressable register of the peripheral system 202.

[0137] Fig. 12 is a flow chart of one embodiment of a method 1200 for obtaining a value stored in an addressable register. The method 1200 may, for example, be embodied within the logic of the peripheral bus interface 206 of Figs. 2 and 9. During a first step 1202 of the method 1200, the address of the addressable register is driven on address signal lines of a bus (e.g., the peripheral bus 204), and an asserted read control signal is driven on a read control signal line of the bus, during a first pipeline stage.

[0138] During a step 1204, the value is received via data signal lines of the bus when a corresponding ready signal driven on a ready signal line of the bus is asserted during a second pipeline stage subsequent to the first pipeline stage.

[0139] Fig. 13 is a flow chart of one embodiment of a method 1300 for providing a value stored in an addressable register. The method 1300 may, for example, be embodied within the logic of the peripheral system 202 of Figs. 2 and 4. During a first step 1302 of the method 1300, an address driven on address signal lines of a peripheral bus is received when a read control signal driven on a read control signal line of a bus (e.g., the peripheral bus 204) is asserted during a first pipeline stage.

[0140] During a step 1304, if the address is an address of the addressable register, the contents of the addressable register are driven on data signal lines of the bus, and an asserted ready signal is driven on a ready signal line of the bus during a second pipeline stage subsequent to the first pipeline stage.

[0141] Fig. 14 is a timing diagram depicting voltages of signals driven on the peripheral bus 204 of Fig. 2 versus time during an exemplary write transaction. During the write transaction of Fig. 14, the CORE_STALL signal is not asserted, and the execution pipeline implemented within the processor 102 of Fig. 2 is not stalled.

[0142] In Fig. 14, the address signal PADR and the asserted write control signal PWR are driven on the corresponding signal lines of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9 during the memory read 0 (M0) stage of the execution pipeline. As described above, the address signal PADR specifies an address of an addressable register of the peripheral system 202 of Fig. 2.

[0143] During the subsequent write back (WB) stage of the execution pipeline, the data signal CDATA and the asserted ready signal CDRDY are driven on the corresponding signal lines of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9. The data signal CDATA is expectedly generated during the execution (EX) stage of the pipeline, and conveys a value to be stored in the addressable register of the peripheral system 202.

[0144] Fig. 15 is a timing diagram depicting voltages of signals driven on the peripheral bus 204 of Fig. 2 versus time during another write transaction. During the write transaction of Fig. 15, the CORE_STALL signal is asserted, and the execution pipeline implemented within the processor 102 of Fig. 2 is stalled for one cycle of the CLOCK signal.

[0145] As in Fig. 14, the address signal PADR and the asserted write control signal PWR are driven on the corresponding signal lines of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9 during the memory read 0 (M0) stage of the execution pipeline. As described above,

the address signal PADR specifies an address of an addressable register of the peripheral system 202 of Fig. 2.

[0146] In Fig. 15, during the subsequent execution (EX) stage of the execution pipeline, a stall condition occurs within the processor core 102 (by a condition other than assertion of the PDRDY signal). As a result, the stall logic 900 of Fig. 9 asserts the CORE_STALL signal during the execution (EX) stage. During the next cycle of the CLOCK signal, the stall condition no longer exists within the processor 102, and the stall logic 900 of Fig. 9 deasserts the CORE_STALL signal. As in Fig. 14, during the subsequent write back (WB) stage of the execution pipeline, the data signal CDATA and the asserted ready signal CDRDY are driven on the corresponding signal lines of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9.

[0147] As a result of the stall condition within the processor core 102, the write data to be sent to the peripheral system 202 is not available in the expected cycle of the CLOCK signal. The peripheral system 202 relies on the CDRDY signal to capture the data signal CDATA during the next cycle of the CLOCK signal. In Fig. 15, the execution (EX) stage of the execution pipeline is effectively extended by one cycle of the CLOCK signal as indicated in Fig. 15. As described above, the data signal CDATA is expectedly generated during the execution (EX) stage of the pipeline, and conveys a value to be stored in the addressable register of the peripheral system 202.

[0148] Fig. 16 is a flow chart of one embodiment of a method 1600 for storing a value in an addressable register. The method 1600 may, for example, be embodied within the logic of the peripheral bus interface 206 of Figs. 2 and 9. During a first step 1602 of the method 1600, the address of the addressable register is driven on address signal lines of a peripheral bus, and an asserted write control signal is driven on a write control signal line of the bus (e.g., the peripheral bus 204) during a first pipeline stage.

[0149] During a step 1604, the value to be stored in the addressable register is driven on data signal lines of the bus, and an asserted ready signal is driven on a ready signal line of the bus, during a second pipeline stage subsequent to the first pipeline stage.

[0150] Fig. 17 is a flow chart of one embodiment of a method 1700 for storing a value in an addressable register. The method 1700 may, for example, be embodied within the logic of the peripheral system 202 of Figs. 2 and 4. During a first step 1702 of the method 1300, an address driven on address signal lines of a bus (e.g., the peripheral bus 204) is received when a write

control signal driven on a write control signal line of the bus is asserted during a first pipeline stage.

[0151] During a step 1704, if the address is an address of the addressable register, a value driven on data signal lines of the bus is received when a corresponding ready signal driven on a ready signal line of the bus is asserted during a second pipeline stage subsequent to the first pipeline stage. The value is stored in the addressable register during a step 1706.

[0152] Fig. 18 is a timing diagram depicting voltages of signals driven on the peripheral bus 204 of Fig. 2 versus time during an exemplary read-modify-write transaction. During the read-modify-write transaction of Fig. 18, the CORE_STALL signal is not asserted, and the execution pipeline implemented within the processor 102 of Fig. 2 is not stalled.

[0153] In Fig. 18, the address signal PADR, the read control signal PDR, and the write control signal PWR are all driven on the corresponding signal lines of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9 during the memory read 0 (M0) stage of the execution pipeline. As described above, the address signal PADR specifies an address of an addressable register of the peripheral system 202 of Fig. 2.

[0154] During the subsequent memory read 1 (M1) stage of the execution pipeline, the data signal PDATA and the asserted ready signal PDRDY are driven on the corresponding signal lines of the peripheral bus 204 is asserted by the peripheral system 202 of Fig.2. The data signal PDATA expectedly conveys a value obtained from the addressable register of the peripheral system 202.

[0155] During the subsequent execution (EX) stage of the execution pipeline, the bit manipulation unit (BMU) 902 of the peripheral bus interface 206 of Fig. 9 expectedly modifies the value obtained from the addressable register of the peripheral system 202 as described above. During the subsequent write back (WB) stage of the execution pipeline, the data signal CDATA and the asserted ready signal CDRDY are driven on the corresponding signal lines of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9. The data signal CDATA expectedly conveys the modified value produced by the bit manipulation unit (BMU) 902 of the peripheral bus interface 206.

[0156] Figs. 19A and 19B in combination form a flow chart of one embodiment of a method 1900 for modifying a value stored in an addressable register. The method 1900 may, for example, be embodied within the logic of the peripheral bus interface 206 of Figs. 2 and 9. During a first

step 1902 of the method 1900, an address of the addressable register is driven on address signal lines of a bus (e.g., the peripheral bus 204), an asserted read control signal is driven on a read control signal line of the bus, and an asserted write control signal is driven on a write control signal line of the bus during a first pipeline stage.

[0157] During a step 1904, a value driven on data signal lines of the bus is received as a value stored in the addressable register when a corresponding first ready signal driven on a ready signal line of the bus is asserted during a second pipeline stage subsequent to the first pipeline stage.

[0158] During a step 1906, the value stored in the addressable register is modified during a third pipeline stage subsequent to the second pipeline stage.

[0159] During a step 1908, the modified value is driven on data signal lines of the bus, and an asserted ready signal is driven on a ready signal line of the bus, during a fourth pipeline stage subsequent to the third pipeline stage.

[0160] Fig. 20 is a timing diagram depicting voltages of signals driven on the peripheral bus 204 of Fig. 2 versus time during an exemplary interrupt request. During the interrupt request of Fig. 20, the interrupt inhibit signal IRQINH is not asserted.

[0161] In Fig. 20, the asserted interrupt request signal PIRQ and the corresponding interrupt vector signal PIVECT are driven on the corresponding signal lines of the peripheral bus 204 by the interrupt control unit 400 of the peripheral system 202 of Fig. 4 during a cycle of the CLOCK signal. During the next cycle of the CLOCK signal, the logic of the instruction sequencing unit 302 of Figs. 8 and 9 pushes the program counter onto the stack of the trap program counter (TPC) unit 802 of Figs. 8 and 9, and stores an address of a first instruction of an interrupt service routine associated with the interrupt request in the register of the program counter (PC) unit 800 of Figs. 8 and 9 reserved for the program counter. During the subsequent cycle of the CLOCK signal, the processor 102 starts fetching instructions of an interrupt service routine associated with the first interrupt request.

[0162] In Fig. 20, the interrupt return instruction of the interrupt service routine is grouped for execution during the grouping (GR) stage of the execution pipeline, and the interrupt return signal IRQRET is asserted by the logic of the instruction sequencing unit 302 of Figs. 8 and 9 as described above.

[0163] During the subsequent operand read (RD) stage of the execution pipeline, the interrupt return signal IRQRET is driven on the corresponding signal line of the peripheral bus 204 by the

peripheral bus interface 206 of Fig. 9 as described above. During the operand read (RD) stage, the logic of the instruction sequencing unit 302 of Figs. 8 and 9 pops the stack of the trap program counter (TPC) unit 802 of Figs. 8 and 9 and stores the resultant value in the register of the program counter (PC) unit 800 of Figs. 8 and 9 reserved for the program counter as described above. This value is the address of the next instruction of the program that was interrupted by the first interrupt request. The processor 102 starts fetching and executing instructions of the interrupted program during the next cycle of the CLOCK signal.

[0164] It is noted that the timing diagrams of Figs. 10-11, 14-15, and 18-20 reflect the embodiment of the peripheral bus interface 206 and the load/store unit 304 of Fig. 9. In the embodiment of Fig. 9, the decode logic 904 decodes the native opcodes during the operand read (RD) stage, the address generation unit 906 produces the address of the addressable register during the address generation (AG) stage, and the peripheral bus interface 206 drives the address signal PADR on the corresponding signal lines of the peripheral bus 204 during the memory read 0 (M0) stage. Other embodiments of the peripheral bus interface 206 and/or the load/store unit 304 are possible and may have different corresponding timing diagrams.

[0165] Fig. 21 is a timing diagram depicting voltages of signals driven on the peripheral bus 204 of Fig. 2 versus time during an exemplary nested interrupt request. During the nested interrupt request of Fig. 21, the interrupt inhibit signal IRQINH is not asserted, and while a first interrupt request is being handled, a second interrupt request is received, wherein the first interrupt request has a higher priority than the second interrupt request.

[0166] In Fig. 21, the asserted interrupt request signal PIRQ and the corresponding interrupt vector signal PIVECT are driven on the corresponding signal lines of the peripheral bus 204 by the interrupt control unit 400 of the peripheral system 202 of Fig. 4 during a cycle of the CLOCK signal. During the next cycle of the CLOCK signal, the logic of the instruction sequencing unit 302 of Figs. 8 and 9 pushes the program counter onto the stack of the trap program counter (TPC) unit 802 of Figs. 8 and 9, and stores an address of a first instruction of an interrupt service routine associated with the first interrupt request in the register of the program counter (PC) unit 800 of Figs. 8 and 9 reserved for the program counter.

[0167] During the subsequent cycle of the CLOCK signal, the processor 102 starts fetching instructions of the interrupt service routine associated with the first interrupt request. During the same cycle of the CLOCK signal, the asserted new interrupt request signal PNIRQ and the

corresponding interrupt vector signal PIVECT associated with the second interrupt request are driven on the corresponding signal lines of the peripheral bus 204 by the interrupt control unit 400 of the peripheral system 202 of Fig. 4. As the second interrupt request has a lower priority than the first interrupt request, the logic of the instruction sequencing unit 302 of Fig. 8 pushes an address of a first instruction of an interrupt service routine associated with the second interrupt request on the stack of the trap program counter (TPC) unit 802 of Figs. 8 and 9 and continues to execute the instructions of the interrupt service routine associated with the first interrupt request.

[0168] The interrupt return instruction of the interrupt service routine associated with the first interrupt request is grouped for execution during the grouping (GR) stage of the execution pipeline, and the interrupt return signal IRQRET is asserted by the logic of the instruction sequencing unit 302 of Figs. 8 and 9 as described above. During the subsequent operand read (RD) stage of the execution pipeline, the asserted interrupt return signal IRQRET is driven on the corresponding signal line of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9. This first asserted interrupt return signal IRQRET driven on the peripheral bus 204 is associated with the interrupt signal PIRQ (i.e., with the first interrupt request) as indicated in Fig. 21.

[0169] During the operand read (RD) stage labeled "RD #1" in Fig. 21, the logic of the instruction sequencing unit 302 of Figs. 8 and 9 pops the stack of the trap program counter (TPC) unit 802 of Figs. 8 and 9 and stores the resultant value in the register of the program counter (PC) unit 800 of Figs. 8 and 9 reserved for the program counter as described above. This value is the address of the first instruction of the interrupt service routine associated with the second interrupt request, and the processor 102 starts fetching and executing instructions of the interrupt service routine associated with the second interrupt request during the next cycle of the CLOCK signal.

[0170] The interrupt return instruction of the interrupt service routine associated with the first interrupt request is grouped for execution during the grouping (GR) stage of the execution pipeline, and the interrupt return signal IRQRET is asserted by the logic of the instruction sequencing unit 302 of Figs. 8 and 9 as described above. During the subsequent operand read (RD) stage of the execution pipeline, the asserted interrupt return signal IRQRET is driven on the corresponding signal line of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9. This second asserted interrupt return signal IRQRET driven on the peripheral bus 204 is associated with the interrupt signal PNIRQ (i.e., with the second interrupt request) as indicated in Fig. 21.

[0171] During the operand read (RD) stage labeled "RD #2" in Fig. 21, the logic of the instruction sequencing unit 302 of Figs. 8 and 9 pops the stack of the trap program counter (TPC) unit 802 of Figs. 8 and 9 and stores the resultant value in the register of the program counter (PC) unit 800 of Figs. 8 and 9 reserved for the program counter as described above. This value is the address of the next instruction of the program that was interrupted by the first interrupt request. The processor 102 starts fetching and executing instructions of the interrupted program during the next cycle of the CLOCK signal.

[0172] Fig. 22 is a timing diagram depicting voltages of signals driven on the peripheral bus 204 of Fig. 2 versus time during another nested interrupt request. During the nested interrupt request of Fig. 22, the interrupt inhibit signal IRQINH is not asserted, and while a first interrupt request is being handled, a second interrupt request is received, wherein the first interrupt request has a lower priority than the second interrupt request.

[0173] In Fig. 22, the asserted interrupt request signal PIRQ and the corresponding interrupt vector signal PIVECT are driven on the corresponding signal lines of the peripheral bus 204 by the interrupt control unit 400 of the peripheral system 202 of Fig. 4 during a cycle of the CLOCK signal. During the next cycle of the CLOCK signal, the logic of the instruction sequencing unit 302 of Figs. 8 and 9 pushes the program counter onto the stack of the trap program counter (TPC) unit 802 of Figs. 8 and 9, and stores an address of a first instruction of an interrupt service routine associated with the first interrupt request in the register of the program counter (PC) unit 800 of Figs. 8 and 9 reserved for the program counter. During the subsequent cycle of the CLOCK signal, the processor 102 starts fetching instructions of the interrupt service routine associated with the first interrupt request.

[0174] Two cycles of the CLOCK signal later, the asserted interrupt request signal PIRQ and the corresponding interrupt vector signal PIVECT associated with the second interrupt request are driven on the corresponding signal lines of the peripheral bus 204 by the interrupt control unit 400 of the peripheral system 202 of Fig. 4. As the second interrupt request has a higher priority than the first interrupt request, the logic of the instruction sequencing unit 302 of Fig. 8 pushes an address of a next instruction of the first interrupt service routine on the stack of the trap program counter (TPC) unit 802 of Figs. 8 and 9, and stores an address of a first instruction of an interrupt service routine associated with the second interrupt request in the register of the program counter (PC) unit 800 of Figs. 8 and 9 reserved for the program counter. During the subsequent cycle of

the CLOCK signal, the processor 102 starts fetching instructions of the interrupt service routine associated with the second interrupt request.

[0175] The interrupt return instruction of the interrupt service routine associated with the second interrupt request is grouped for execution during the grouping (GR) stage of the execution pipeline, and the interrupt return signal IRQRET is asserted by the logic of the instruction sequencing unit 302 of Figs. 8 and 9 as described above. During the subsequent operand read (RD) stage of the execution pipeline, the asserted interrupt return signal IRQRET is driven on the corresponding signal line of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9. This first asserted interrupt return signal IRQRET driven on the peripheral bus 204 is associated with the second interrupt signal PIRQ (i.e., with the second interrupt request) as indicated in Fig. 22.

[0176] During the operand read (RD) stage labeled "RD #2" in Fig. 22, the logic of the instruction sequencing unit 302 of Figs. 8 and 9 pops the stack of the trap program counter (TPC) unit 802 of Figs. 8 and 9 and stores the resultant value in the register of the program counter (PC) unit 800 of Figs. 8 and 9 reserved for the program counter as described above. This value is the address of the next instruction of the interrupt service routine associated with the first interrupt request, and the processor 102 starts fetching and executing instructions of the interrupt service routine associated with the first interrupt request again during the next cycle of the CLOCK signal.

[0177] The interrupt return instruction of the interrupt service routine associated with the first interrupt request is grouped for execution during the grouping (GR) stage of the execution pipeline, and the interrupt return signal IRQRET is asserted by the logic of the instruction sequencing unit 302 of Figs. 8 and 9 as described above. During the subsequent operand read (RD) stage of the execution pipeline, the asserted interrupt return signal IRQRET is driven on the corresponding signal line of the peripheral bus 204 by the peripheral bus interface 206 of Fig. 9. This second asserted interrupt return signal IRQRET driven on the peripheral bus 204 is associated with the first interrupt signal PIRQ (i.e., with the first interrupt request) as indicated in Fig. 22.

[0178] During the operand read (RD) stage labeled "RD #1" in Fig. 21, the logic of the instruction sequencing unit 302 of Figs. 8 and 9 pops the stack of the trap program counter (TPC) unit 802 of Figs. 8 and 9 and stores the resultant value in the register of the program counter (PC) unit 800 of Figs. 8 and 9 reserved for the program counter as described above. This value is the address of the next instruction of the program that was interrupted by the first interrupt request.

The processor 102 starts fetching and executing instructions of the interrupted program during the next cycle of the CLOCK signal.

[0179] Figs. 23A and 23B in combination form a flow of one embodiment of a method 2300 for handling an interrupt request. The method 2300 may be, for example, embodied within the interrupt control unit 400 of Fig. 4. During a first step 2302 of the method 2300, an asserted first interrupt request signal is received (e.g., from one of the peripheral devices 404 of Fig. 4). During a step 2304, a second interrupt request signal (e.g., the interrupt request signal PIRQ of Figs. 20-22) is asserted, and the asserted second interrupt request signal and a priority value of the first interrupt request signal are driven on corresponding signal lines of a bus (e.g., the corresponding interrupt signal lines of the peripheral bus 204 of Figs. 2-4 and 8).

[0180] During a step 2306, an asserted third interrupt request signal is received after the first interrupt request signal is received (e.g., from one of the peripheral devices 404 of Fig. 4). During a decision step 2308, a priority value of the third interrupt request signal is compared to the priority value of the first interrupt request signal. If the priority value of the third interrupt request signal is greater than that the priority value of the first interrupt request signal, a step 2310 is performed. On the other hand, if the priority value of the third interrupt request signal is less than that the priority value of the first interrupt request signal, a step 2312 is performed.

[0181] During the step 2310, the second interrupt request signal (e.g., the interrupt request signal PIRQ of Figs. 20-22) is asserted, and the asserted second interrupt request signal and the priority value of the third interrupt request signal are driven on corresponding signal lines of the bus. (See Fig. 22 and the above description of Fig. 22.)

[0182] During the step 2312, a fourth interrupt request signal (e.g., the new interrupt request signal PNIRQ of Figs. 20-22) is asserted, and the asserted fourth interrupt request signal and the priority value of the third interrupt request signal are driven on corresponding signal lines of the bus. (See Fig. 21 and the above description of Fig. 21.)

[0183] Fig. 24 is a diagram of one embodiment of a data processing system 2400, wherein the data processing system 2400 is one embodiment of the data processing system 100 of Fig. 1. In the data processing system 2400, the processor 102 is coupled to, and in communication with, a coprocessor 2402 via a coprocessor bus 2404. As indicated in Fig. 24, the processor 102 includes a coprocessor interface 2406 adapted for coupling to signal lines of the coprocessor bus 2404.

[0184] In general, the processor 102 and the coprocessor 2402 cooperate to achieve a desired result. For example, as described in detail below, the coprocessor 2402 may extend or augment a computational capability of the processor 102. Alternately, or in addition, steps of a computational algorithm may be divided among the processor 102 and the coprocessor 2402. For example, computationally demanding steps of an algorithm may be assigned to the coprocessor 2402, relieving the processor 102 of the need to perform the computationally demanding steps. In many cases, a performance of the data processing system 2400 including the processor 102 and the coprocessor 2402 exceeds a performance of the processor 102 alone.

[0185] In the embodiment of Fig. 24, the coprocessor bus 2404 includes several signal lines conveying signals between the processor 102 and the coprocessor 2402, including a 1-bit "SYNC" signal, an n-bit "COMMAND" signal ($n > 1$), a 1-bit "VALID" signal, a 32-bit "SOURCEA" signal, a 32-bit "SOURCEB" signal, a 32-bit "RESULT" signal.

[0186] In general, the SYNC signal indicates whether the processor 102 expects the coprocessor 2402 to produce a result, and to provide the result via the RESULT signal, within a certain amount of time. The n-bit COMMAND signal specifies an n-bit, user-defined command, and is provided by the processor 102 and the coprocessor 2402. The user-defined command includes multiple ordered bits, wherein the values of the bits are assigned by a user. In general, the coprocessor 2402 is configured to interpret the user-defined command specified by the n-bit COMMAND signal, and to perform a corresponding function. Performance of the corresponding function may, for example, produce the result.

[0187] The VALID signal indicates whether the n-bit COMMAND signal is valid. The 32-bit SOURCEA and SOURCEB signals convey data from the processor 102 to the coprocessor 2402. For example, in response to a valid n-bit COMMAND signal, the coprocessor 2402 may perform a function on data conveyed by the 32-bit SOURCEA and SOURCEB signals, thereby producing a result. The RESULT signal is used to convey a result produced by the coprocessor 2402 to the processor 102.

[0188] In the embodiment of Fig. 24, the coprocessor interface 2406 of the processor 102 generates the SYNC signal, the n-bit COMMAND signal, the VALID signal, and the SOURCEA and SOURCEB signals, and receives the RESULT signal from the coprocessor 2402.

[0189] As indicated in Fig. 24, the coprocessor 2402 may also produce a 1-bit "STALL" signal received by the processor 102. The coprocessor 2402 may assert the STALL signal when a

previous SYNC signal indicated the processor 102 expects the coprocessor 2402 to provide a result via the RESULT signal within a certain amount of time, and the coprocessor 2402 is not able to provide the result within the allotted amount of time. In this situation, the coprocessor 2402 may continue to assert the STALL signal until the result is produced. As indicated in Fig. 24, the processor 102 receives the STALL signal via the coprocessor interface 2406.

[0190] In the embodiment of Fig. 24, the processor 102 is coupled to the memory system 110. As described above, in general, the processor 102 fetches and executes instructions of a predefined instruction set stored in the memory system 110. As illustrated in Fig. 24, the memory system 110 includes a software program (i.e., code) 112 including instructions from the instruction set. The code 112 includes a coprocessor (COP) instruction 2408 of the instruction set.

[0191] As described in detail below, the coprocessor instruction 2408 includes a user-defined command directed to the coprocessor 2402. The user-defined command includes multiple ordered bits having values assigned by the user. During execution of the coprocessor instruction 2408, the processor 102 provides the user-defined command to the coprocessor 2402. In response to the user-defined command, the coprocessor 2402 performs a predetermined function.

[0192] In "tightly coupled" embodiments of the data processing system 2400, the coprocessor 2402 may depend on the processor 102 to access the memory system 110 and to provide data from the memory system 110 to the coprocessor 2402. In other "loosely coupled" embodiments of the data processing system 2400, the coprocessor 2402 may be coupled to the memory system 110 as indicated in Fig. 24, and may access the memory system 110 directly.

[0193] In the loosely coupled embodiments of the data processing system 2400, the processor 102 typically does not expect the coprocessor 2402 to produce a result within a certain amount of time. In this situation, the coprocessor 2402 may assert an "INTERRUPT" signal when the coprocessor 2402 to produces the result. In response to the INTERRUPT signal, the processor 102 may obtain the result from the coprocessor 2402 (e.g., via the RESULT signal) as described in detail below.

[0194] The processor 102 may be, for example, one of several functional blocks or units (i.e., "cores") formed on an integrated circuit. It is now possible for integrated circuit designers to take highly complex functional units or blocks, such as processors, and integrate them into an integrated circuit much like other less complex building blocks.

[0195] Fig. 25 is a diagram of one embodiment of the processor 102 of Fig. 24. As indicated in Fig. 2, the processor 102 receives the above described clock signal CLOCK and executes instructions dependent upon the CLOCK signal. Mores specifically, the processor 102 includes several functional units described below, and operations performed within the functional units are synchronized by the CLOCK signal. The processor 102 also receives other interrupt signals (e.g., from devices other than the coprocessor 2402).

[0196] In the embodiment of Fig. 25, in addition to the coprocessor interface 2406 of Fig. 24, the processor 102 includes the following functional units shown in Fig. 3 and described above: the instruction prefetch unit 300, the instruction sequencing unit 302, the load/store unit (LSU) 304, the execution unit 306, the register files 308, and the pipeline control unit 310.

[0197] In the embodiment of Fig. 25, the processor 102 is a pipelined superscalar processor core. That is, the processor 102 implements the instruction execution pipeline of Fig. 6 and described above. The instruction execution pipeline of Fig. 6 includes multiple pipeline stages, concurrently executes multiple instructions in different pipeline stages, and is also capable of concurrently executing multiple instructions in the same pipeline stage.

[0198] In general, as described above, the pipeline control unit 310 controls the instruction execution pipeline. In the embodiment of Fig. 25, the pipeline control unit 310 includes an interrupt control unit 2500. The interrupt control unit 2500 receives the INTERRUPT signal from the coprocessor 2402 of Fig. 24, and interrupt signals from other devices as indicated in Fig. 25.

[0199] In general, the interrupt control unit 2500 implements a vectored priority interrupt system in which higher priority interrupts are handled (i.e., serviced) first. A non-maskable interrupt (NMI) signal has the highest priority of all the interrupt signals. In one embodiment, the interrupt control unit 2500 includes a 16-bit interrupt request register having bit locations corresponding to 2 non-maskable interrupt signals and 14 maskable interrupt bit locations. The 2 non-maskable interrupt signals include the NMI signal and a device emulation interrupt (DEI) signal. When an interrupt signal is received, the corresponding bit location in the interrupt request register is set to '1'. Each bit location in the interrupt request register is cleared only when the processor 102 services the corresponding interrupt signal, or explicitly by software.

[0200] In one embodiment, the interrupt control unit 2500 also includes an interrupt mask register containing mask bit locations for each of the 14 maskable interrupts. A mask bit value of '0' (i.e., a cleared bit) prevents the corresponding interrupt from being serviced (i.e., masks the

corresponding interrupt signal). The INTERRUPT signal may be one of the 14 maskable interrupt signals.

[0201] In one embodiment, the interrupt control unit 2500 also includes two 16-bit interrupt priority registers. Consecutive bit locations in each of the interrupt priority registers are used to store user-defined priority levels associated with the 14 maskable interrupt signals. Software programs may write to the bit locations of the interrupt priority registers. User-defined interrupt priorities may range from 0b00 (i.e., decimal '0') to 0b11 (i.e., decimal '3'), with 0b00 being the lowest and 0b11 being the highest. (The NMI signal has a fixed priority level of decimal '5', and the DEI signal has a fixed priority level of decimal '4'.)

[0202] Once the interrupt control unit 2500 decides to service an interrupt, the interrupt control unit 2500 signals the instruction sequencing unit 302 of Fig. 25 to stop grouping instructions in the grouping (GR) stage of the execution pipeline. Instructions fetched and partially decoded up to and including those in the grouping (GR) stage are flushed. Executions of instructions in the operand read (RD) stage, the address generation (AG) stage, the memory access 0 (M0) stage, the memory access 1 (M1) stage, and the execution (EX) stage are completed normally before instructions of the service routine are fetched and executed.

[0203] In one embodiment, the instruction set executable by the processor 102 of Fig. 24 includes two special types of instructions facilitating communication between the processor 102 and the coprocessor 2402: a "CPCOM" instructions and "CPOUT" instructions. The coprocessor instruction 2408 of Fig. 24 may be, for example, a CPCOM instruction or a CPOUT instruction. In general, the CPCOM instructions are used to obtain a result from the coprocessor 2402 via the RESULT signal within a certain amount of time. More specifically, the CPCOM instructions are used to obtain a result from the coprocessor 2402 via the RESULT signal during pipeline execution of the CPCOM instruction as described in more detail below. Certain CPCOM instructions described below may be used to both provide data to the coprocessor 2402 via the SOURCEA and SOURCEB signals, and to obtain a result from the coprocessor 2402 via the RESULT signal during pipeline execution of the CPCOM instruction.

[0204] The CPOUT instructions, on the other hand, are generally used to provide data to the coprocessor 2402 of Fig. 24 via the SOURCEA and SOURCEB signals of Figs. 24 and 25. The CPOUT instructions might be used, for example, in a loosely-coupled embodiment of the data processing system 2400 of Fig. 24. As described above, in such loosely coupled embodiments, the

coprocessor 2402 may assert the INTERRUPT signal of Figs. 24 and 25 when the coprocessor 2402 produces the result. In response to the INTERRUPT signal, the interrupt control unit 2500 of Fig. 25 may initiate execution of a corresponding interrupt service routine within the processor 102 of Figs. 24 and 25. The interrupt service routine may include a CPCOM instruction that obtains the result from the coprocessor 2402 via the RESULT signal.

[0205] Figs. 26A-27C illustrate exemplary embodiments of the coprocessor instruction 2408 of Fig. 24, wherein the coprocessor instruction 2408 is a CPCOM instruction or a CPOUT instruction. In the embodiments of Figs. 26A-27C, the register files 308 of Fig. 25 includes an address register file and a general purpose register file. The address register file includes 8 32-bit address registers, and the general purpose register file includes 16 16-bit general purpose registers. The 16 16-bit registers of the general purpose register file can be paired to form 8 32-bit general purpose registers. Each of the 16 16-bit general purpose registers can be specified using 3 bits, and each of the 8 32-bit address registers and the 8 32-bit general purpose registers can be specified using 3 bits.

[0206] Fig. 26A is a diagram of one embodiment of the coprocessor instruction 2408 of Fig. 24, wherein the coprocessor instruction 2408 is a CPCOM instruction. In the embodiment of Fig. 26A, the coprocessor instruction 2408 includes an opcode field 2602, a destination register field 2604, a source register 1 field 2606, a source register 2 field 2608, and an 8-bit user command field 2610.

[0207] The opcode field 2602 contains a value identifying the instruction as a CPCOM instruction, and specifying the particular embodiment of the coprocessor instruction 2408 of Fig. 26A. The destination register field 2604 specifies a register of the register files 308 of Fig. 25 into which a result produced by the coprocessor 2402 of Fig. 24 and conveyed by the RESULT signal is to be saved.

[0208] The source register 1 field 2606 specifies a register of the register files 308 of Fig. 25 containing data to be sent to the coprocessor 2402 of Fig. 24 via the SOURCEA signal. The source register 2 field 2608 specifies another register of the register files 308 containing data to be sent to the coprocessor 2402 via the SOURCEB signal.

[0209] The 8-bit user command field 2610 is used to hold an 8-bit, user-defined command to be sent to the coprocessor 2402 via the COMMAND signal of Figs. 24 and 25 (n = 8). In the embodiment of Fig. 26A, the user-defined command includes 8 ordered bits, the values of which

are assigned by the user. During execution of the coprocessor instruction 2408 of Fig. 26A by the processor 102 of Fig. 24, the coprocessor interface 2406 of Fig. 24 drives the 8 bits of the user command field 2610 on 8 corresponding signal lines conveying the COMMAND signal from the processor 102 to the coprocessor 2402.

[0210] Fig. 26B is a diagram of another embodiment of the coprocessor instruction 2408 of Fig. 24, wherein the coprocessor instruction 2408 is a CPCOM instruction. In the embodiment of Fig. 26B, the coprocessor instruction 2408 includes an opcode field 2612, a source/destination register field 2614, a source register field 2616, and a 16-bit user command field 2618.

[0211] The opcode field 2612 contains a value identifying the instruction as a CPCOM instruction, and specifying the particular embodiment of the coprocessor instruction 2408 Fig. 26B. The source/destination register field 2614 both: (i) specifies a register of the register files 308 of Fig. 25 containing data to be sent to the coprocessor 2402 of Fig. 24 via the SOURCEA signal, and (ii) specifies a register of the register files 308 of Fig. 25 into which a result produced by the coprocessor 2402 of Fig. 24 and conveyed by the RESULT signal is to be saved. The source register field 2616 specifies another register of the register files 308 containing data to be sent to the coprocessor 2402 via the SOURCEB signal.

[0212] The 16-bit user command field 2618 is used to hold a 16-bit, user-defined command to be sent to the coprocessor 2402 via the COMMAND signal of Figs. 24 and 25 (n = 16). In the embodiment of Fig. 26B, the user-defined command includes 16 ordered bits, the values of which are assigned by the user. During execution of the coprocessor instruction 2408 of Fig. 26B by the processor 102 of Fig. 24, the coprocessor interface 2406 of Fig. 24 drives the 16 bits of the user command field 2618 on 16 corresponding signal lines conveying the COMMAND signal from the processor 102 to the coprocessor 2402.

[0213] Fig. 26C is a diagram of a third embodiment of the coprocessor instruction 2408 of Fig. 24, wherein the coprocessor instruction 2408 is a CPCOM instruction. In the embodiment of Fig. 26C, the coprocessor instruction 2408 includes an opcode field 2620, a destination register field 2622, and a 16-bit user command field 2624. The coprocessor instruction 2408 of Fig. 26C is used to obtain a result from the coprocessor 2402 of Fig. 24 via the RESULT signal.

[0214] The opcode field 2620 contains a value identifying the instruction as a CPCOM instruction, and specifying the particular embodiment of the coprocessor instruction 2408 Fig. 26C. The destination register field 2622 specifies a register of the register files 308 of Fig. 25 into which

a result produced by the coprocessor 2402 of Fig. 24 and conveyed by the RESULT signal is to be saved.

[0215] The 16-bit user command field 2624 is used to hold a 16-bit, user-defined command to be sent to the coprocessor 2402 via the COMMAND signal of Figs. 24 and 25 (n = 16). In the embodiment of Fig. 26C, the user-defined command includes 16 ordered bits, the values of which are assigned by the user. During execution of the coprocessor instruction 2408 of Fig. 26C by the processor 102 of Fig. 24, the coprocessor interface 2406 of Fig. 24 drives the 16 bits of the user command field 2624 on 16 corresponding signal lines conveying the COMMAND signal from the processor 102 to the coprocessor 2402.

[0216] Fig. 27A is a diagram of one embodiment of the coprocessor instruction 2408 of Fig. 24, wherein the coprocessor instruction 2408 is a CPOUT instruction. In the embodiment of Fig. 27A, the coprocessor instruction 2408 includes an opcode field 2702, a source register 1 field 2704, a source register 2 field 2706, and a 16-bit user command field 2708.

[0217] The opcode field 2702 contains a value identifying the instruction as a CPOUT instruction, and specifying the particular embodiment of the coprocessor instruction 2408 Fig. 27A. The source register 1 field 2704 specifies a register of the register files 308 of Fig. 25 containing data to be sent to the coprocessor 2402 of Fig. 24 via the SOURCEA signal. The source register 2 field 2706 specifies another register of the register files 308 containing data to be sent to the coprocessor 2402 via the SOURCEB signal.

[0218] The 16-bit user command field 2708 is used to hold an 16-bit, user-defined command to be sent to the coprocessor 2402 via the COMMAND signal of Figs. 24 and 25 (n = 16). In the embodiment of Fig. 27A, the user-defined command includes 16 ordered bits, the values of which are assigned by the user. During execution of the coprocessor instruction 2408 of Fig. 27A by the processor 102 of Fig. 24, the coprocessor interface 2406 of Fig. 24 drives the 16 bits of the user command field 2708 on 16 corresponding signal lines conveying the COMMAND signal from the processor 102 to the coprocessor 2402.

[0219] Fig. 27B is a diagram of another embodiment of the coprocessor instruction 2408 of Fig. 24, wherein the coprocessor instruction 2408 is a CPOUT instruction. In the embodiment of Fig. 27B, the coprocessor instruction 2408 includes an opcode field 2710, a source register field 2712, and a 16-bit user command field 2714.

[0220] The opcode field 2710 contains a value identifying the instruction as a CPOUT instruction, and specifying the particular embodiment of the coprocessor instruction 2408 Fig. 27B. The source register field 2712 specifies a register of the register files 308 containing data to be sent to the coprocessor 2402 via the SOURCEA signal.

[0221] The 16-bit user command field 2714 is used to hold a 16-bit, user-defined command to be sent to the coprocessor 2402 via the COMMAND signal of Figs. 24 and 25 (n = 16). In the embodiment of Fig. 27B, the user-defined command includes 8 ordered bits, the values of which are assigned by the user. During execution of the coprocessor instruction 2408 of Fig. 27B by the processor 102 of Fig. 24, the coprocessor interface 2406 of Fig. 24 drives the 16 bits of the user command field 2714 on 16 corresponding signal lines conveying the COMMAND signal from the processor 102 to the coprocessor 2402.

[0222] Fig. 27C is a diagram of a third embodiment of the coprocessor instruction 2408 of Fig. 24, wherein the coprocessor instruction 2408 is a CPOUT instruction. In the embodiment of Fig. 27C, the coprocessor instruction 2408 includes an opcode field 2716 and a 16-bit user command field 2718. The coprocessor instruction 2408 of Fig. 27C is used to send a user-defined command to the coprocessor 2402 of Fig. 24 via the COMMAND signal of Figs. 24 and 25.

[0223] The opcode field 2716 contains a value identifying the instruction as a CPOUT instruction, and specifying the particular embodiment of the coprocessor instruction 2408 Fig. 27C. The 16-bit user command field 2718 is used to hold a 16-bit, user-defined command to be sent to the coprocessor 2402 via the COMMAND signal of Figs. 24 and 25 (n = 16). In the embodiment of Fig. 27C, the user-defined command includes 16 ordered bits, the values of which are assigned by the user. During execution of the coprocessor instruction 2408 of Fig. 27C by the processor 102 of Fig. 24, the coprocessor interface 2406 of Fig. 24 drives the 16 bits of the user command field 2718 on 16 corresponding signal lines conveying the COMMAND signal from the processor 102 to the coprocessor 2402.

[0224] Fig. 28 is a diagram illustrating how operations of the coprocessor 2402 are synchronized with operations of the processor 102 of Figs. 24 and 25 during execution of the coprocessor instruction 2408 of Fig. 24. The execution pipeline of the processor 102, shown in Fig. 6 and described above, includes the operand read (RD), the address generation (AG), the memory address 0 (M0), the memory address 1 (M1), and the execution (EX) stages illustrated in Fig. 6.

[0225] As indicated in Fig. 28, when the coprocessor instruction 2408 provides data to the coprocessor 2402, values stored in registers of the register files 308 of Fig. 25 specified by source register fields of the coprocessor instruction 2408 are obtained during the operand read (RD) pipeline stage, and used to generate the SOURCEA and SOURCEB signals. The SYNC, COMMAND, and VALID signals are also generated during the operand read (RD) pipeline stage. The 1-bit SYNC signals specifies whether the coprocessor instruction 2408 is a CPCOM instruction or a CPOUT instruction.

[0226] At the end of the operand read (RD) stage, the generated SOURCEA, SOURCEB, SYNC, COMMAND, and VALID signals are stored in registers (i.e., "registered") as indicated in Fig. 28, and provided to the coprocessor 2402 at the beginning of the address generation (AG) stage.

[0227] When the coprocessor instruction 2408 of Fig. 24 is a CPCOM instruction, the coprocessor 2402 is expected to generate the RESULT signal before or during the memory address 1 (M1) stage. At the end of the memory address 1 (M1) stage, the RESULT signal produced by the coprocessor 2402 is registered as indicated in Fig. 28, and provided to other logic within the processor 102 at the beginning of the execution (EX) stage. During the execution (EX) stage, the processor 102 stores the result value conveyed by the RESULT signal in a register of the register files 308 of Fig. 25 specified by the destination register field of the coprocessor instruction 2408 (i.e., of the CPCOM instruction).

[0228] When the coprocessor 2402 is expected to generate the RESULT signal before or during the memory address 1 (M1) stage and is not able to do so, the coprocessor 2402 may assert the STALL signal. In response to the STALL signal, the execution pipeline of the processor 102 is stalled (e.g., by the pipeline control unit 310 of Fig. 25). The coprocessor 2402 may continue to assert the STALL signal until the coprocessor 2402 is able to generate the RESULT signal. When the coprocessor 2402 deasserts the STALL signal, the execution pipeline is resumed (e.g., by the pipeline control unit 310), and the processor 102 stores the result value conveyed by the RESULT signal in the register of the register files 308 of Fig. 25 specified by the destination register field of the coprocessor instruction 2408 of Fig. 24 (i.e., of the CPCOM instruction).

[0229] Fig. 29 is a diagram of one embodiment of the data processing system 2400 of Fig. 24 wherein the processor 102 and the coprocessor 2402 are loosely coupled. In the embodiment of Fig. 29, the coprocessor 2402 accesses the memory system 110 directly.

[0230] In the embodiment of Fig. 29, the processor 102 executes a software application program wherein the coprocessor instruction 2408 is a CPOUT instruction. The coprocessor instruction 2408 (i.e., the CPOUT instruction) causes the processor 102 to provide a command and/or data to the coprocessor 2402 via the COMMAND and SOURCEA and SOURCEB signals. In general, the processor 102 does not expect the coprocessor 2402 to produce a result, and to generate the RESULT signal, within a certain amount of time.

[0231] In the embodiment of Fig. 29, when the coprocessor 2402 produces a result, the coprocessor 2402 asserts the INTERRUPT signal. In response to the INTERRUPT signal, the interrupt control unit 2500 of Fig. 25 initiates execution of a corresponding interrupt service routine within the processor 102. The interrupt service routine includes a CPCOM instruction that obtains the result from the coprocessor 2402 via the RESULT signal. It is noted that in the loosely-coupled embodiment of Fig. 29, the STALL signal of Figs. 24, 25, and 28 is not used.

[0232] The particular embodiments disclosed above are illustrative only, as the invention may be modified and practiced in different but equivalent manners apparent to those skilled in the art having the benefit of the teachings herein. Furthermore, no limitations are intended to the details of construction or design herein shown, other than as described in the claims below. It is therefore evident that the particular embodiments disclosed above may be altered or modified and all such variations are considered within the scope and spirit of the invention. Accordingly, the protection sought herein is as set forth in the claims below.